68th International Astronautical Congress (IAC), Adelaide, Australia, 25-29 September 2017.

IAC-17- C1.5.2x40525

Distributed Control Of An Evolving Satellite Assembly During In-Orbit Construction

Rebecca C. Foust[a]*, Michelle Zhao[b], Suzanne Oliver[b], Soon-Jo Chung[b], Fred Y. Hadaegh[c]

[a] *Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, 104 S Wright St, Urbana, IL, USA,* foust3@illinois.edu
[b] *Department of Engineering and Applied Sciences, California Institute of Technology, 1200 E California Blvd, Pasadena, CA, USA,* mzhao@caltech.edu, soliver@caltech.edu, sjchung@caltech.edu
[c] *Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr, Pasadena, CA, USA,* fred.y.hadaegh@jpl.nasa.gov
* Corresponding Author

Abstract

This paper presents a method for controlling sets of docked satellites during in-orbit construction of a large-scale satellite assembly from a swarm of heterogeneous satellites. Such a system can be used to enable missions from sparse aperture telescopes to elaborate space stations. Once two or more agents from the swarm are docked, the resulting assembly is an over-actuated system so position and attitude controllers must determine which of the available actuators to use. Typically, control allocation for over-actuated systems is done using a simple linear program, but for this scheme the mass properties and number of control points changes. As a result, the linear program solved changes with each new agent that docks with the assembly so the agents must know how to alter the linear program for additional agents and remove control points whose plumes would interact with those agents. In most systems, this linear program is solved by a central computer, but for this system the actuators belong to distinct agents so to increase reliability, each agent solves the same linear program and executes its portion of the resulting control command. This paper sets up the general linear program that each agent in the assembly must solve and then establishes the rules for altering that program when new agents dock. Initial simulations allow the agents to dock as they come into proximity along their respective trajectories to their target locations. This can lead to instability and uncontrollability if the agents dock in certain configurations, so the control allocation rules are extended to prevent uncontrollable or unstable docking scenarios. The logic used for this is based on the moment of inertia and the available actuation ability. Simulations in 6DOF perturbed satellite dynamics show the efficacy of this approach in preventing uncontrollable assemblies and bringing the assemblies together into the desired final configuration.

Keywords: Control Allocation, Self-Assembly, Autonomy

Nomenclature

B	Control influence matrix
CG	XYZ position of center of gravity
$\boldsymbol{d}_{act}$	Actuator direction wrt body frame
$\mathbf{f}_{\text{des}}$	Vector of desired XYZ forces and torques (6x1)
I_3	3x3 Identity matrix
${}^{A}J$	Moment of Inertia tensor wrt frame A
m	Mass
N_T	Number of thrusters
${}^{A}\boldsymbol{r}_b$	Vector b wrt frame A
${}^{B}\boldsymbol{r}_{bD}$	Vector from CG of b to dock port in frame B
${}^{A}R^{B}$	Rotation from frame B to frame A
u	Control vector (N_T x1)

Acronyms/Abbreviations

Swarm Orbital Construction Algorithm (SOCA), Fault Detection Identification and Recovery (FDIR), Six Degrees of Freedom (6DOF), Pulse Width Modulated (PWM), Rotations Per Minute (RPM), Robot Operating System (ROS)

1. Introduction

Design and construction of large space systems is often constrained by factors that have more to do with surviving launch than the intended mission, like launch vehicle fairing size or ability to withstand launch loading. Satellites constructed in space would not experience these design constraints, allowing for lighter, more capable satellites. Start to finish construction in orbit is not yet possible, but improvements can still be made through recent advances in swarm spacecraft guidance and control [1,2,3] and autonomous rendezvous and docking [4].

By leveraging the above swarm guidance and control algorithms, a large space structure can be constructed from a swarm of component satellites. The advantages of such a mission are clear: increased reliability due to redundancy, increased flexibility, ability to reconfigure for future missions, and ability to self-repair [5]. Applications for such missions range from the small scale, where the components are microsatellites building a support structure for a distributed telescope or a solar sail, to the large scale, where components are habitat modules building a space colony. At any scale, the guidance and control problem

of assembly remains the same. A sample mission is illustrated in Figure 1, using two types of agents. The rod agent is a rectangular prism with two docking ports and the connector agent is a hexagonal prism with six docking ports. The mission steps are as follows:

Step 1. The components enter into loose formation to stay close to other components until they are used (e.g. collision-free J_2-invariant passive relative orbits [2]).

Step 2. The components determine their desired final position in the assembly and move to take the position using SOCA.

Step 3. Along the path to the final position, components assigned to neighboring positions dock and proceed combined.

Step 4. Finally, a complete structure is made once all components have reached their final destination.

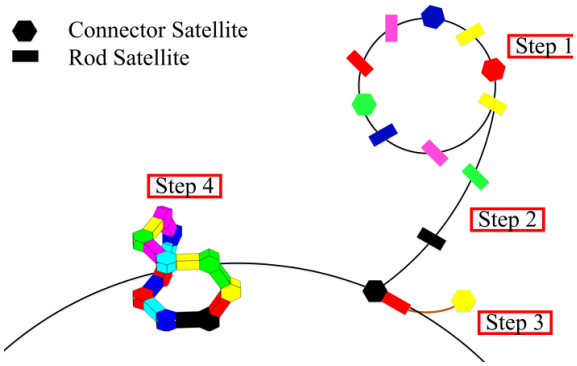


Fig. 1 Outline of Sample Mission

Previous work on this problem resulted in the creation of SOCA. SOCA takes an arbitrary number of agents and a set of desired final positions and generates the optimal trajectories to those final positions in a 6DOF spacecraft dynamical environment while avoiding unwanted collisions [6,7]. Through the execution of SOCA, the agents in the swarm dock along their trajectories to their desired final locations in the assembly. This is not realistic, because once the agents are docked, they should be treated as a single, new agent with different mass properties and thus a different control trajectory. The new assembly will be over-actuated, which means there are now more control points than there are degrees of freedom in the system.

This paper will focus on the control allocation problem as the agents are docking. With each new agent docked, the control allocation problem changes because new control points are added and the overall system mass properties have changed. Though control allocation of over-actuated systems is very well studied, very few teams have looked into control allocation for an increasing number of control points.

1.1 Related Work

Control allocation for over-actuated systems can be done in a variety of ways. Typically the method used depends on the level of fidelity required and the computational abilities of the system. The simplest way to allocate control is the pseudo-inverse method, which involves solving Problem 1 for the control input vector:

$$\min_{\mathrm{u}} \frac{1}{2}\boldsymbol{u}^T\boldsymbol{u} \text{ s.t. } B\boldsymbol{u} = \boldsymbol{f}_{des} \quad (1)$$

This leads to a closed form solution of $u = B^T(BB^T)\boldsymbol{f}_{des}$, where $B^T B\, B^T$ is referred to as the pseudoinverse. A pseudoinverse is necessary for over-actuated systems since the number of linearly independent columns of B is greater than the number of rows of B [8]. This method is somewhat disadvantageous since it does not allow for the injection of other constraints, like minimum and maximum control effort.

Another method for control allocation which can account for these additional constraints is to establish a linear or quadratic optimization problem, then find an efficient optimization solver. Different cost functions can be leveraged to achieve different objectives, such as minimizing total control effort, minimizing error in the control trajectory, and minimizing power used by the actuators. The cost functions can also be combined as long as they remain consistent with the program type. Problem 2 is an example of such a linear program, designed to minimize the error in the allocation.

$$\min_{\mathbf{u}} \|B\boldsymbol{u} - \boldsymbol{f}_{des}\|_1 \quad \begin{bmatrix}\boldsymbol{u}\\ -\boldsymbol{u}\end{bmatrix} \geq \begin{bmatrix}u_{min}\\ -u_{max}\end{bmatrix} \quad (2)$$

These types of solvers allow for a significant increase in complexity of the problem solved, but have a corresponding increase in computation time which is highly dependent on the solver used.

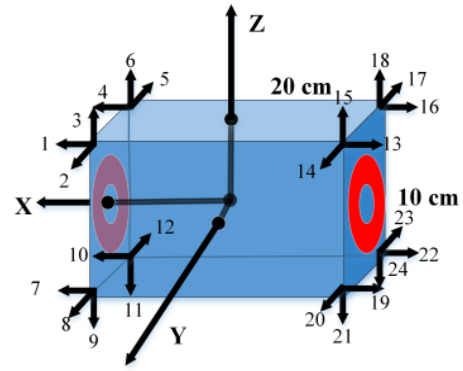
Several adaptations exist to allocate control in the case of actuator failure, which is the crux the recovery portion of FDIR [9]. In this case however, actuators must be added and the mass properties must be changed. This has been most commonly handled in previous docked satellites through gain scheduling, where the controller gains are pre-determined for each configuration and stored in a table [10,11]. This is disadvantageous for an on-orbit assembly scheme because the sheer number of gains to be computed and stored is intractable. Another common method is system identification, where the docked spacecrafts characterize the mass properties by actuating thrusters and calculating the response. It is fuel and time intensive to generate high-fidelity models. The final option is online model calculation, in which each agent stores its mass properties and as they dock, they calculate the new mass properties of the combined system [10]. This is advantageous because it does not require much data storage or any fuel usage, but it can be sensitive to errors in docking alignment.

Very few sources in literature have looked into this online model calculation problem and solutions

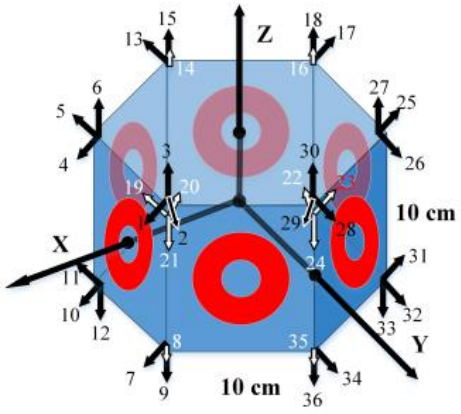
typically end at determining the new mass and control properties to use with the controller and neglect to automate the removal of blocked actuators from the allocation problem [12].

1.2 *Shape Parameters*

The rod agent is a rectangular prism with two docking ports located on the ends. The connector agent is a regular hexagonal prism with six docking ports along the sides. In order to execute this algorithm while incorporating attitude control, assumptions about mass properties and systems engineering configurations for the agent types must be made. The mass and volume advantages of the swarm will be most effective if the agents are kept small, in the nanosatellite class.



a)



b)

Fig. 2 Definitions of the Rod and Connector agent types, with docking ports shown in red

The rod agent uses a standard 2U CubeSat bus. Each side of the regular hexagonal prism must be the same as the face of a CubeSat to allow docking. The size and shape parameters for the two agent types are listed below in Table 1. The body frame definitions for the two agent types are shown in Fig. 2, along with the location, direction, and numbering of the thrusters. Optimization of the thruster quantity, positions, and directions is left to future work.

Table 1. Mass Properties of the two agent types

Type	Units	Rod	Connector
Mass	kg	2.67	4
I_{xx}	kg-cm^2	44.5	116.7
I_{yy}	kg-cm^2	111.3	116.7
I_{zz}	kg-cm^2	111.3	166.7

This paper describes an online model aggregation and control allocation method which combines mass properties and removes actuators blocked by docking. This method will demonstrate the ability to prevent uncontrollable docks when used in conjunction with SOCA.

2. Material and Methods

This work involves simulations done in Matlab making extensive use of the program CVX, a Matlab package for specifying and solving convex programs [13,14]. The simulation in Matlab uses the SOCA to generate collision-free assembly trajectories in a J_2 perturbed spacecraft dynamical environment [4,6,7], then uses the work of this paper to allocate the control for the assembly as the various agents dock along their trajectories.

A separate simulation was created using the maximum efforts required over a SOCA trajectory to determine which potential docking configurations were infeasible by examining the control allocation matrix at these configurations and the maximum control effort available to the thrusters.

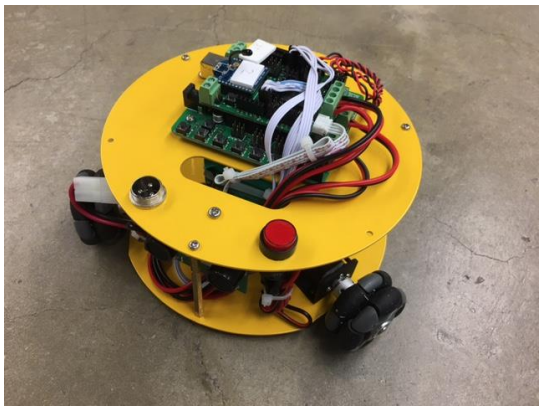


Fig. 3: Omni-Directional 3-wheeled robot used to experimentally validate SOCA.

Experiments were also performed to test the validity of SOCA and to pave the way for tests of SOCA and the control allocation algorithm on a high-fidelity spacecraft simulator which is currently under construction at Caltech. NEXUS 3-Wheeled Compact Omni-Directional Arduino Compatible Mobile Robots (shown in Fig. 3) were used for this experiment. The robots are controlled using Arduinos with XBee communication devices. The SOCA is run on a separate computer through the use of ROS. This computer uses ROS to access the Vicon Motion Capture system, run the algorithm, and send commands to the robots. This computer then communicates commands to the robots

through the Xbee serial wireless modules attached to each of the robots. SOCA generates trajectories which require Vicon feedback to follow, so each robot is given the relevant Vicon information, position and orientation, along with the trajectory waypoint. Onboard the robot, a PD controller is used to generate the desired wheel RPMs, then map those values to the actual robot wheel commands, a set of three PWM values. A six-robot system was tested in a 2-meter-by-2-meter motion capture space, where each robot read in its trajectory, given by a list of way-points provided by an offline system SOCA algorithm.

Multiple preliminary tests were performed to characterize the relationship between the input PWM signal and the motor RPM. Inconsistencies were observed in the RPM at a constant PWM even on smooth surfaces. This problem was ameliorated by closing the loop on-board the robots using the motor encoders, which gives better control over the speed and path of the robot. The encoders are read at each control loop (about 0.1s) and the PWM is corrected based on the encoder readings

3. Theory and Calculation

Combining the agent types above into any feasible structure requires a modularized approach to updating the model. This can be done by requiring that only one dock be performed at any given time. To effectively combine two models into one, we must first define a set of information that each model must contain:

- System mass
- Center of gravity
- Moment of inertia tensor
- Control influence matrix

We assume that each component satellite has its center of gravity at the geometric center of the object. Combining the masses and determining the new center of gravity is straightforward:

$$m_{ay} = m_1 + m_2 \tag{3}$$

$${}^1CG_{ay} = \frac{m_2}{m_{ay}}[{}^1r_{1D} + {}^1r_{2D}] \tag{4}$$

From here the moment of inertia tensor can be calculated using the parallel axis theorem:

$${}^1J_{ay,CG} = {}^1J_1 + m_1\left(\left({}^1r_{1,CG}\right)^2 I_3 - {}^1\boldsymbol{r}_{1,CG}\,{}^1\boldsymbol{r}_{1,CG}{}^T\right) + {}^1R^2\,{}^2J_2({}^1R^2)^T + m_2\left(\left({}^1r_{2,CG}\right)^2 I_3 - {}^1\boldsymbol{r}_{2,CG}\,{}^1\boldsymbol{r}_{2,CG}{}^T\right) \tag{5}$$

Before calculating the changes to the control influence matrix, the control influence matrix of each agent type must be calculated.

$$B_2 = \begin{bmatrix} B_{Force} \\ B_{Torque} \end{bmatrix} = \begin{bmatrix} {}^2[\boldsymbol{d}_{act,1}, \boldsymbol{d}_{act,2}, \dots] \\ {}^2\boldsymbol{r}_{act} \times B_{Force} \end{bmatrix} \tag{6}$$

This can then be transformed for each of the two docking components as follows:

$$B_{2_{ay}} = \begin{bmatrix} {}^1R^2\,{}^2[\boldsymbol{d}_{act,1}, \boldsymbol{d}_{act,2}, \dots] \\ \left({}^1R^2\,{}^2\boldsymbol{r}_{act} + {}^1\boldsymbol{r}_{2,CG} - {}^1CG\right) \times B_{Force} \end{bmatrix} \tag{7}$$

$$B_{ay} = \left[B_{1_{ay}}, B_{2_{ay}}\right] \tag{8}$$

When a new agent is added, the position of the actuators in the A frame must be recalculated before solving for the B matrix.

The new control authority matrix still needs some tweaking to remove actuators that have been blocked by the dock, or whose plumes would interact with other parts of the spacecraft. Actuators can be removed by creating an identity matrix of the size of the number of actuators, then zeroing out the row corresponding to the blocked actuator. This matrix then post-multiplies B to create the effective B matrix, B_{eff}. To determine which actuators are blocked, the configuration of the agents is leveraged. For both agent types, the blocked thrusters can easily be determined if the docking port in use is found. The thrusters associated with each docking port are predetermined and stored, and are considered blocked if the docking port is in use.

A useful extension here is finding the actuators whose plumes would interact with the assembled structure. Finding these actuators involves approximating the plumes as cones and checking for collision between these cones and the assembly. This is a simplistic model of thruster plumes, typically the three-dimensional characteristics of the plumes are taken into consideration and this may be addressed in future work. Collision checking methods from the field of robotics can be leveraged to efficiently determine if the plume cone interacts with the rest of the structure [15,16].

After the unusable thrusters are pruned from the control allocation matrix, the control authority of this potential assembly must be checked. The control authority of the assembly is determined by calculating the singular value decomposition of the B matrix. This is then multiplied by the maximum achievable thrust to determine the maximum $\mathbf{f}_{des}$ that this assembly can achieve. For CubeSat scale actuators, a reasonable value for the maximum thrust of each actuator is 50 mN [17]. If this $\mathbf{f}_{des}$ is sufficient to complete the trajectory, the docking is allowed to occur, if not the agents must not dock and must avoid collision until one or both assemblies have docked with other agents.

While the agents are docked, the control allocation is performed using the following linear program:

$$\begin{gathered} \min_{\mathbf{u}} \sum_i u_i \\ \begin{bmatrix} \boldsymbol{u} \\ -\boldsymbol{u} \end{bmatrix} \geq \begin{bmatrix} 0 \\ -\boldsymbol{u}_{max} \end{bmatrix} \\ B\boldsymbol{u} = \boldsymbol{f}_{des} \end{gathered} \tag{9}$$

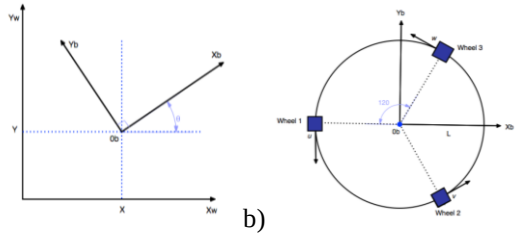
where the B and **u** change for different assemblies as the number of control points change. This program

minimizes the control effort exerted by all actuators subject to constraints on the maximum and minimum values.

For the experiments, a simple PD controller was designed to track the given trajectories. The controller in equation 10 determines the three wheel RPMs based on the error in position, velocity angular position and angular velocity.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = K_p \begin{bmatrix} \sin\theta & -\cos\theta & L \\ \sin(\frac{\pi}{3}-\theta) & \cos(\frac{\pi}{3}-\theta) & L \\ -\sin(\frac{\pi}{3}+\theta) & \cos(\frac{\pi}{3}+\theta) & L \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} - K_d \begin{bmatrix} \sin\theta & -\cos\theta & L \\ \sin(\frac{\pi}{3}-\theta) & \cos(\frac{\pi}{3}-\theta) & L \\ -\sin(\frac{\pi}{3}+\theta) & \cos(\frac{\pi}{3}+\theta) & L \end{bmatrix} \begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} \quad (10)$$

The definitions of the robot frames and wheel positions



a) b)

Fig. 4. Wheeled Robot Frame Definitions

are shown in Fig. 4. In addition to this controller determining the desired RPM, there is also a mapping between RPM and PWM. This mapping was determined heuristically by measuring the RPM at a sampling of PWMs and fitting a curve to this data. This was done for one wheel on each robot, but due to performance issues seen later in the paper, we believe we will need to do this for all wheels.

4. Results and Discussion

4.1 *Control Allocation*

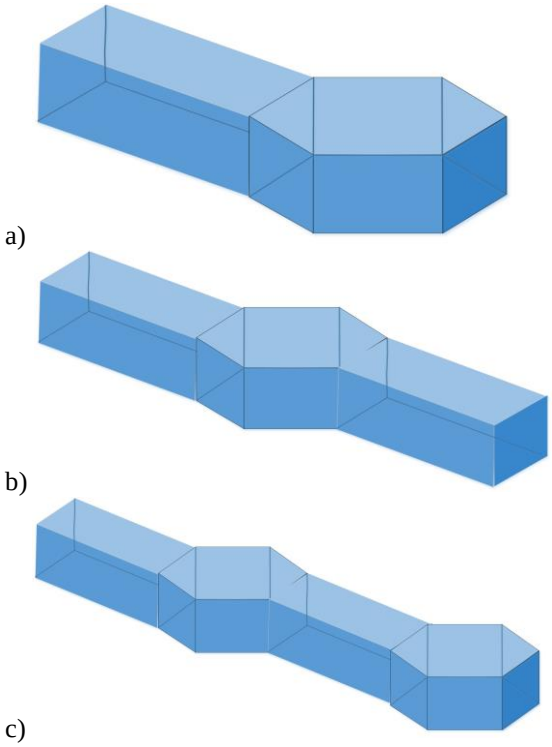
A simple simulation was made using the maximum desired force and torque generated through an execution of the SOCA. This $\mathbf{f}_{des}$ was used to test three docking scenarios: agent to agent docking, agent to assembly docking, and assembly to assembly docking. The error in the allocation was determined as follows:

$$u_{err} = \|B\boldsymbol{u} - \boldsymbol{f}_{des}\|_1 \quad (10)$$

4.1.1 *Agent to Agent Docking*

This test involved a rod agent and a connector agent docking to form an assembly. The resulting shape can be seen in Fig. 5a. A common-sense test was performed to see that each component of $\mathbf{f}_{des}$ was achieved with the correct combination of thrusters.

This assembly was then used to allocate the maximum $\mathbf{f}_{des}$ over a SOCA trajectory as given above. The error in actuation is 2.89e-10 N with a total force of 0.138N using 16 thrusters of the available 52. One problem that is noted in using linear programs to allocate control is the inability to specify an impulse bit, the smallest thrust possible for the controller to produce. As a result, several agents are commanded to give very small thrusts, on the order of 1e-10. This is not achievable with the chosen thruster. To reduce these erroneous micro-fires, we will add the number of thrusters firing to the cost to be minimized. Nearly all of the control error is a result of these micro-fires.



a)

b)

c)

Fig.5: Description of the three basic control allocation tests run before incorporating the algorithm into SOCA

4.1.2 *Agent to Assembly Docking*

The agent to assembly docking involves a connector agent docking with an assembly of a rod and a connector as seen in Fig. 5b. This and the more complex docks must be treated carefully, particularly in the calculation of $r_{2,CG}$, which becomes more complicated the more agents are involve. We circumvent this problem by calculating the value for each docking port upon assembly so that new assemblies can simply sum the vectors. For this assembly, there are 96 thrusters with 16 blocked, giving 80 usable thrusters. The control allocation algorithm uses 22 thrusters a total force of 0.128N with a maximum error of 4.82e-10N.

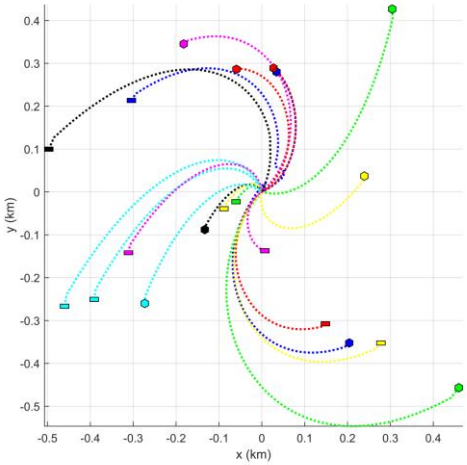


Fig. 6 Results of Control Allocation in SOCA for 10 rod and 10 connector agents targeted to a folded hexagon shape. Zoom view shows the trajectories combining after a dock occurs

4.1.3 *Assembly to Assembly Docking*

The assembly to assembly docking involves two assemblies of a rod and a connector as seen in Fig. 5c. For this assembly there are 120 thrusters with 24 blocked, leaving 96 thrusters. The maximum SOCA $\mathbf{f}_{des}$ is achieved using 40 thrusters at a total force of 0.129 N with a maximum error of 3.16e-10 N.

4.2 *Incorporation Into SOCA*

The model update and control allocation were incorporated into the SOCA to test the calculations over the course of a complete, 20 agent mission. Once agents enter within a predefined distance from each other, they determine if they will dock based on their assignment. If they are intended to dock, now they check the control authority of the assembly they would create. If it is sufficient, they dock. Otherwise, they avoid collision until a later time.

4.2.2 *Multi-Agent 6DOF Simulation*

The results of this simulation can be seen in Fig. 6. Because of the optimality design of SOCA and the scales involved in these small spacecraft docking scenarios, the control allocation algorithm does not come into play until the final time step of the simulation at which point all spacecraft dock into the final structure. The final structure is controllable and has 460 unblocked thrusters, which means the control allocation and model update algorithm does function properly. This issue with SOCA exists because the agents start at a 1km separation and dock at a 20cm separation. The fuel optimality aspect of SOCA means that they will only combine along the way to the final position if their paths are already very close to each other. In future work we will modify the cost function used in the optimal trajectory generation algorithm to emphasize the need to dock before the final time and de-prioritize the fuel cost. This will lead to sub-optimal fuel performance, but will be significantly less risky from a systems standpoint since the docking can be enforced to happen one at a time and the assembly will have time to recover from perturbations caused by docking before the next dock. Another issue with SOCA is that after the dock, the agents are treated as one and the final locations are combined into one. This is not ideal since it removes the target from other possible agents that could benefit from reassigning to that location. In future work this will be fixed by tweaking the assignment algorithm to take into consideration all of the sub-agents in the assembly, which will bid as one for each of the terminal positions.

4.3 *Experimental Validation of SOCA*

In this section, we test the behaviour of physical robots following the SOCA algorithm for optimal trajectory. The main focus is to investigate the performance of SOCA trajectories on actual multi-agent robotic hardware. The experimental results show SOCA performing assignment and trajectory generation for 6 agents in planar final configuration with realistic 3DOF trajectories.

Fig.7 Initial configuration of 6 omni-directional robots for SOCA experiment

The off-board control computer ran the SOCA algorithm given intended starting points of each of the six robots and determined the optimal trajectory with collision avoidance. The starting positions of the robots are shown in Fig. 7 and the time lapse of the full test is shown in Fig 8.The algorithm worked very well however the on-board controller on the robots could not track the trajectories sufficiently well to enable docking. The tracking error of the robots is approximately 10 cm, which is sufficient to eliminate the collision avoidance effects of SOCA. This causes the two rectangular agents to get stuck at t=5-10sec and the bottom rectangular agent to miss its final orientation. To within the tracking error, all of the robots follow the trajectories well. Further investigation is needed to conclusively

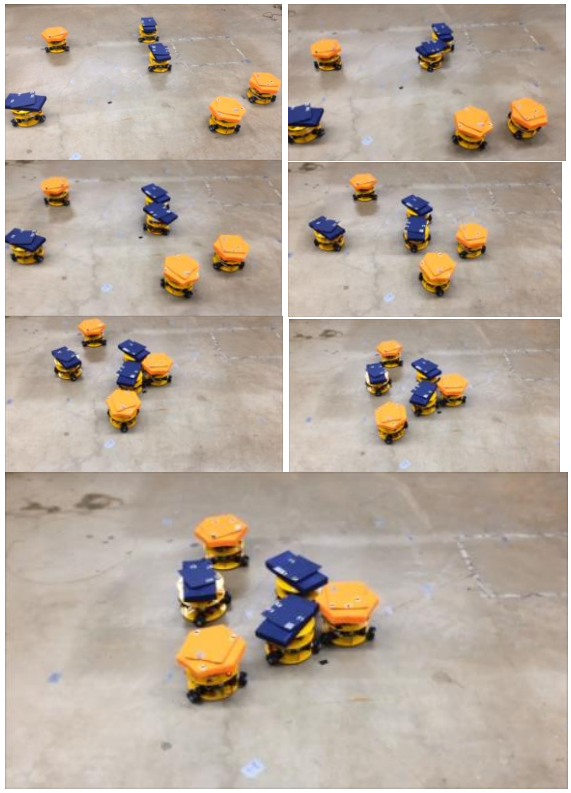


Fig 8 Time lapse of SOCA robot experiment at 5 second intervals

determine the source of the error but the culprit is most likely the low-level motor controller which has very inconsistent performance, as discussed earlier. The intended trajectory is plotted in Fig. 9 with the actual trajectories achieved by the robots. For the most part, each of the robots achieves the correct direction of travel, but the tracking error prevents the trajectories from lining up perfectly. The wiggle seen in blue at the bottom of the figure is likely cause by this error in wheel actuation. The feedback keeps pulling the robot back towards the desired trajectory but the wheel errors keep making it deviate. We will be fixing this error and continuing to perform more tests to validate the algorithm.

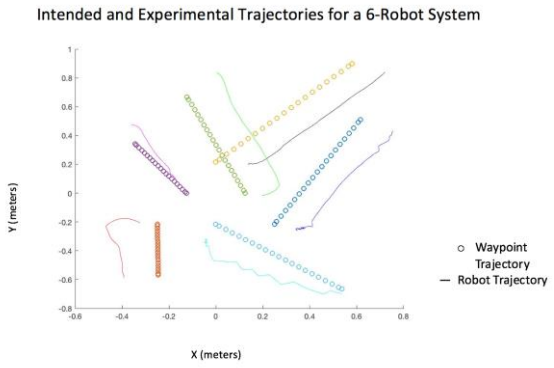


Fig 9 Actual vs desired robot trajectories. The paths taken by each robot from start to end position (line) lined up with the intended linear waypoints (dotted), with some slight offset due to VICON.

In future work, on-board trajectory generation in real-time on a large ground robot system will be implemented to allow for dynamic control and a novel docking apparatus, applicable to ground and space swarms, will be developed and tested on these ground robots to reinforce the swarm configuration.

6. Conclusions

A flexible model update and control allocation algorithm was developed and tested through simulation. Experimental results were also shown which establish the groundwork needed for testing the control allocation algorithm on a set of spacecraft simulators. The control allocation algorithm successfully distributes the control effort over a set of test structures and throughout a simulation of the SOCA with a 20-agent assembly. In addition to control allocation, the algorithm also evaluates the feasibility of potential docks, which allows the SOCA to prevent docks which would make the assembly uncontrollable.

Acknowledgements

This work was supported by a NASA Space Technology Research Fellowship. Government sponsorship is acknowledged. This research was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the NASA.

References

[1] F.Y. Hadaegh, S.-J. Chung, and H.M. Manohara. On development of 100-gram-class spacecraft for swarm applications. IEEE Systems Journal, 10(2):673-684, 2016.

[2] D. Morgan, S.-J. Chung, L. Blackmore, B. Acikmese, D. Bayard, and F.Y. Hadaegh. Swarm-keeping Strategies for spacecraft under J2 and atmospheric drag perturbations. Journal of Guidance, Control, and Dynamics, 35(5):1492-1506, 2012.

[3] D. Morgan, S.-J. Chung, and F.Y. Hadaegh. Model predictive control of swarms of spacecraft using sequential convex programming. Journal of Guidance, Control, and Dynamics, 37(6):1725-1740, 2014.

[4] W. Fehse. Automated rendezvous and docking of spacecraft, volume 16. Cambridge university press, 2003.

[5] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G.S. Chirikjian. Modular self-reconfigurable robot systems. IEEE

Robotics & Automation Magazine, 14(1):43-52, 2007.

[6] R.C. Foust, S.-J. Chung, and F.Y. Hadaegh. Autonomous in-orbit satellite assembly from a modular heterogeneous swarm using sequential convex programming. In AIAA/AAS Astrodynamics Specialist Conference, page 5271, 2016.

[7] R. C. Foust, S.-J. Chung, and F.Y Hadaegh. Real-time optimal control and targe.t assignment for autonomous in-orbit satellite assembly from a modular heterogeneous swarm. In AIAA/AAS Space Flight Mechanics Meeting, 2016.

[8] M.W. Oppenheimer, D.B. Doman, and M.A. Bolender. "Control allocation for over-actuated systems." Control and Automation, 2006. MED'06. 14th Mediterranean Conference on. IEEE, 2006.

[9] N. Swain, and S. Manickavasagar. "A combined fault detection, identification and reconfiguration system based around optimal control allocation." Fault Tolerant Flight Control. Springer Berlin Heidelberg, 2010. 399-422.

[10] Mohan, Swati. "Tools for Reconfigurable Control System Comparisons for Autonomous Assembly Applications." International Astronautical Congress, Daejon, South Korea, IAC-09 C. Vol. 1. 2009.

[11] Jewison, Christopher Michael. Reconfigurable thruster selection algorithms for aggregative spacecraft systems. Diss. Massachusetts Institute of Technology, 2014.

[12] Mohan, Swati. Quantative selection and design of model generation architectures for on-orbit autonomous assembly. Diss. Massachusetts Institute of Technology, 2010.

[13] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx, September 2013.

[14] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs, Recent Advances in Learning and Control (a tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, editors, pages 95-110, Lecture Notes in Control and Information Sciences, Springer, 2008. http://stanford.edu/~boyd/graph_dcp.html.

[15] H.M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, Principles of robot motion: theory, algorithms, and implementation. MIT press, 2005.

[16] F. Baldini, S. Bandyopadhyay, R. Foust, S.-J. Chung, A. Rahmani, J.-P. de la Croix, A. Bacula, C.M. Chilan, and F.Y. Hadaegh. Fast motion planning for agile space systems with multiple obstacles. In AIAA/AAS Astrodynamics Specialist Conference, page 5683, 2016.

[17] CubeSat Propulsion Systems: CubeSat Propulsion Systems Overview, 1 September 2017, http://www.cubesat-propulsion.com/vacco-systems/